Optimizing Weight Mapping and Data Flow for Convolutional Neural Networks on RRAM based Processing-In-Memory Architecture

Xiaochen Peng[1], Rui Liu[2], and Shimeng Yu[1]
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA[1]
School of Electrical, Computing and Energy Engineering, Arizona State University, Tempe, AZ[2]
Email: shimeng.yu@ece.gatech.edu

***Abstract*— Resistive random access memory (RRAM) based array architecture has been proposed for on-chip acceleration of convolutional neural networks (CNNs), where the array could be configured for dot-product computation in a parallel fashion by summing up the column currents. Prior processing-in-memory (PIM) designs unroll each 3D kernel of the convolutional layers into a vertical column of a large weight matrix, where the input data will be accessed multiple times. As a result, significant latency and energy are consumed in interconnect and buffer. In this paper, in order to maximize both weight and input data reuse for RRAM based PIM architecture, we propose a novel weight mapping method and the corresponding data flow which divides the kernels and assign the input data into different processing-elements (PEs) according to their spatial locations. The proposed design achieves ~65% save in latency and energy for interconnect and buffer, and yields overall 2.1× speed up and ~17% improvement in the energy efficiency in terms of TOPS/W for VGG-16 CNN, compared with the prior design based on the conventional mapping method.**

Keywords— non-volatile memory, processing-in-memory, machine learning, deep neural network, hardware accelerator

I. INTRODUCTION

The state-of-the-art deep neural networks (DNNs) have achieved remarkable success in various applications, including speech recognition and image classification. To avoid the extensive off-chip data access during weighted-sum and weight-update, it is desirable to design the DNN hardware accelerators which can efficiently implement the entire algorithms on-chip to achieve significant speed-up and power reduction. To achieve this goal, the embedded non-volatile memory such as RRAM is of great interests.

Recently, lots of efforts have been made to design silicon CMOS ASIC accelerators that could utilize the distributed computations in an array of multiply-accumulate (MAC) units with local registers and global buffers (such as TPU [1] and Eyeriss [2]), while the computation is still performed in the digital domain. A more energy-efficient approach is the processing-in-memory (PIM) where the computation is embedded into the memory directly, by analog computation with the column current summation [3]. In such context, the crossbar array with RRAM is an ideal platform for dot-product computation in DNNs. As the two-terminal selector technology is premature, one-transistor-one-resistor (1T1R) based pseudo-crossbar is practical for large-scale integration [3]. RRAM could store multi-bit weights by exploiting the multi-conductance-state, which makes it attractive as "analog" synapses with higher density. Fig. 1 shows the pseudo-crossbar array that can naturally perform "analog" matrix-vector multiplication, where the word lines (WLs) could be turned on simultaneously for dot-product computation. The input vector is represented as voltage inputs of the bit-lines (BLs), and the dot-product value will be the current passing through the RRAM cells that sharing one sense-line (SL), such that the sum of dot-product in each column can be obtained by an analog-to-digital converter (ADC) at the end of each SL. Recently, several RRAM based PIM accelerators for DNNs (e.g. ISAAC [5], PRIME [6], etc.) have been proposed.

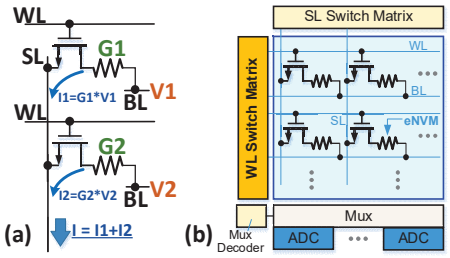

Fig. 1. (a) Using a sense-line (SL) current to perform sum of dot-products. (b) Pesudo-crossbar array with peripherical circuits Switch Matrix, MUX+MUX Decoder and ADC.

To implement the convolutional layers, a basic weight mapping method is to unroll each 3D kernel into a long vertical column. Since the depth of input and output channels could be large (hundreds by hundreds), using a single large matrix to implement one convolutional layer may cause slow-access and extra energy consumption, thus, array partitioning [7] can be introduced to parallelize the computation into multiple sub-arrays. However, the input data will be reused repeatedly for convolution, and we aim to solve the unnecessary latency and energy waste in interconnect and buffer due to this input data reuse in this work. Here we focus on a RRAM based PIM architecture that supports 8-bit weight and 8-bit activation for CNN inference, which exploits a novel weight mapping and data flow to maximize both the input and weight data reuse. We benchmark the hardware performance using a circuit-level macro model NeuroSim [8] at 32 nm CMOS node and compare with a design based on the conventional mapping method.

II. BACKGROUND

Fig. 2 shows how the convolutional layer computes the outputs: In layer<n+1>, the size of input feature maps (IMFs)

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

is W×W×D (D is the depth of input feature channel), which are the outputs from layer<n>, and the kernel size is K×K×D×N (D is the kernel depth), considering same-padding and the stride equals to one, the output feature maps (OFMs) of layer<n+1> will be W×W×N (N is the depth of output feature channel). From the example shown in Fig. 2, it is clear that the kernels slide over the input data to perform elementwise multiplications with a certain stride, and then sum up the partial sums to get the output, which means part of the input data will be reused for the computation.

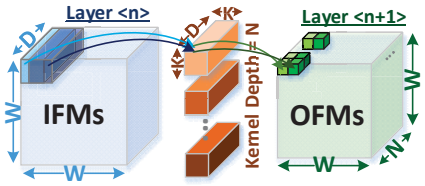

Fig. 2. Example of convolutional layer computation.

With the PIM approach, the kernels (weight) will be mapped into memory crossbar arrays as conductance of each cross-point, such that, this 3D elementwise multiplication will be transformed to a dot-product multiplication. Since the partial sums in each 3D kernel will be summed up to get the final output, it is straightforward to unroll each 3D kernel into a long vertical column, by utilizing the nature of crossbar array which could perform the sum of dot-products in the SLs. In this way, all the kernels in each convolutional layer is mapped into a large weight matrix. Fig. 3 shows the basic weight mapping method [9] (or baseline in this work). With the same example in Fig. 2, one 3D kernel with size K×K×D could be unrolled to a long vertical column which length equals to K×K×D, the kernel depth is N, which means there are N vertical columns in total. Thus, in layer<n+1>, the kernels could be mapped to a large weight matrix, whose length and width equals to K×K×D and N, respectively.

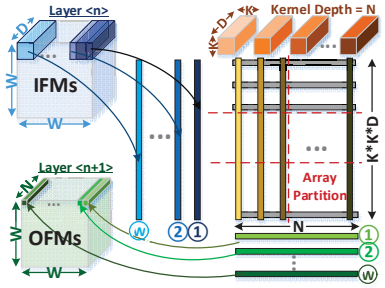

Fig. 3. A basic mapping method of input and weight data, with kernal moving in multiple cycles [10].

To get the OFMs, as Fig. 3 shows, at first cycle, a part of IFMs (shown in dark blue cube) will be multiplied with each 3D kernels, the sum of dot-products from the first kernel will be the first element in the first OFM, the sum of dot-products from the second kernel will be the first element in the second OFM, and so on, thus, at the first cycle, we could get the first elements in every OFM (shown in light green row in size 1×1×N); in the same way, at the second cycle, the kernels will "slide over" the inputs with a stride (equals to one in this example), it is clear that part of the IMFs used in the first cycle will be reused at the second cycle, to generate the second elements in every OFM. Thus, to generate the total OFMs in layer<n+1>, we need to "slide over" the IFMs by W×W times, i.e. we need W×W cycles to finish the computation.

Typically, the kernel size varies in different convolutional layers, thus the unrolled weight matrix size is quite different, which leads to various number of sub-arrays representing different layers. Hence, it is impractical to reuse the unrolled input data along various number of sub-arrays, since it needs different control signals to send each segment of input data to several sub-arrays. Therefore, it is crucial to design a novel mapping method and data flow that could maximize input data reuse, where the weight data and input data can be mapped according to their spatial location, and the hardware can actually "slide over" the input data with a global control unit.

III. PIM Architecture Design Based on Novel Weight Mapping and Data Flow

A. A Novel Weight Mapping Method

In this work, we propose a novel mapping method as Fig. 4. Instead of unrolling 3D kernels into a large matrix, we map the weights at different spatial location of each kernel into different sub-matrices. Hence, K×K sub-matrices are needed for the K×K kernels, and the input data at each kernel location will be sent to the corresponding sub-matrix, respectively.

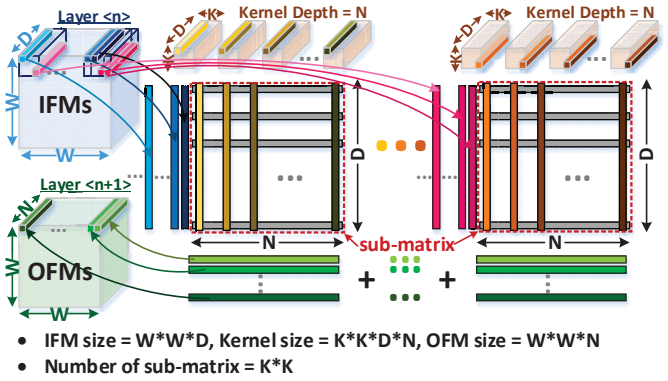

Fig. 4. A novel mapping method that map the weights along the spatial location to a group of sub-matrix.

Each sub-matrix can be represented by a group of sub-arrays which makes the sub-matrix to be large enough to hold the kernels from various layers, as the Fig. 6 (c) shows, such group of sub-arrays with the necessary input and output buffers and accumulation modules can be defined as a processing element (PE). One should be noted that array partitioning within the PE is helpful to maximize the memory utilization. Since the kernels from some convolutional layers (normally the first couple of layers) could be shallow and small, which will not fill the entire PE and cause memory waste. With a group of partitioned sub-arrays, those shallow kernels could be duplicated in different sub-arrays and take multiple input data to generate independent outputs simultaneously. In this case,

some convolutional layers which have shallow kernels but large IFMs could speed up significantly.

B. Data Flow to Maximize IFM Reuse

With this novel mapping method, which divides the kernels and assign the input data into different PEs according to their spatial locations, it is possible to maximize the reuse of input data. Fig. 5 shows an example of processing a 3×3 kernel. At the very beginning, all the input data are at the corresponding PEs, i.e. at T=1, IMF[1][1] (a vector with length D) is assigned to PE[1][1], IMF[1][2] is assigned to PE[1][2], and so on, the partial sums in these 9 PEs will be summed up (by adder tree) to get OFM[2][2] (with size 1×1×N). At the next cycle, the IFMs that will be used for the next computation will be transferred to the neighboring PEs and the useless IFMs will be released, i.e. at T=2, IFM[1][1] is released, IFM[1][2] is transferred from PE[1][2] to PE[1][1], IFM[2][2] is transferred from PE[2][2] to PE[2][1], and so on, then the partial sums in these 9 PEs will give OFM[2][3]. By passing the used IMFs in the same direction as the kernel "slides over" the inputs, the IMFs can be used among PEs efficiently.

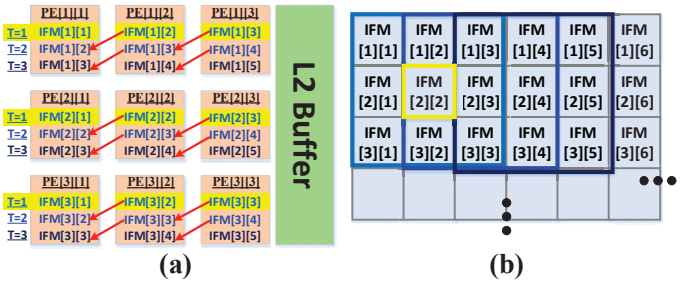

Fig. 5. (a) An example of IFMs transferred among PEs in mutiple cycles. (b) An example of how the kernel "slides over" the input in mutiple cycles.

C. RRAM based PIM Architecture

Fig. 6 (a) shows the top-level RRAM based PIM architecture, which contains multiple tiles, accumulation units, pooling and activation units, L3 buffer and the global control,. Fig. 6 (b) shows a tile contains multiple PEs with routers and L2 buffer. Within a tile, the routers make it possible to communicate among PEs and transfer partial sums from PEs to accumulation units. Fig. 6 (c) shows a PE contains multiple sub-arrays and L1 buffer. Fig. 6 (d) shows the modules within one sub-array. To implement 8-bit analog synaptic weights, we use 4 RRAM cells (assuming each RRAM could store 2-bit weight), the 8-bit fixed-point neuron activations are mapped as 8-bit sequential cycles with input voltages. The shift-add modules are used to shift and accumulate the partial sums of the 8-bit sequential input voltages during 8 cycles. The sub-array size is set to 128×128, to guarantee maximum memory efficiency. Since in most shallow convolutional layers, the depth of feature channels could be smaller than 128, thus, if we use larger sub-array size (i.e. 256×256), at least half of the memory will not be used. Also, we do not further downsize our sub-array (to 64×64), because normally the periphery dominates the area of sub-array, and thus degrade the area efficiency (i.e. comparing four 64×64 sub-arrays with one 128×128 sub-array, the area efficiency of the latter is higher).

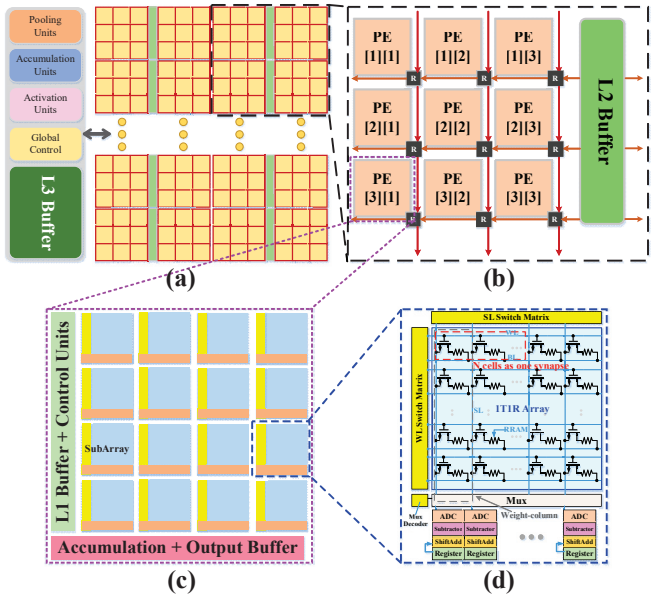

Fig. 6. The diagram of (a) RRAM based PIM architecture; (b) a tile with multiple PEs, routers and L2 buffer; (c) a PE contains 16 sub-arrays, L1 buffer and control units, accumulation modules and output buffer; (d) parallel 8-bit RRAM sub-array.

IV. Results and Discussion

A. Simulations Setup

To evaluate the hardware performance of the RRAM based PIM architecture, we utilized a circuit-level macro model NeuroSim [8] to estimate the area, latency and energy based on at 32nm CMOS node where RRAM integration is feasible from the industry's perspective. We use a relatively large-scale weight (as used in TPU [1]), to guarantee the sufficient computation accuracy. We set the basic weight mapping method with conventional H-tree routing as the baseline, to evaluate the latency and energy saving in interconnect and buffer in this work. Table I shows the component parameters on chip. The energy is given in energy per unit (or module and PE) per operation (or bit), i.e. sub-array energy is 25.04 nJ/op, which tells the total energy for one single sub-array to do one vector-matrix multiplication (or dot-product computation).

B. Results and Discussion

According to the novel weight mapping and data flow in this work, instead of passing the entire K×K IFMs, every time the PE groups only takes the new 1×K IFMs from L2 buffer, and the used (K−1)×K IFMs will be moved forward to neighboring PEs and be reused for the computation of next OFMs. Hence, compared to the baseline, only a small amount of bits will be visited in buffers, and interconnect will transfer much less data every time in this work. Here we consider the latency and energy for one CIFAR image inference. Due to the limitation of bus width and the number of bits that can be read out at each access for different buffer size, around 30% of the total latency and 23% of the total energy consumption are caused by the interconnect and buffers according to the simulation result.

TABLE I. CHIP PARAMETERS

Component	Spec.		Energy	Area (mm^2)
Chip Level				
PE	Number	432	0.202 µJ/PE/op	54.864
L3 Buffer	Device	SRAM	0.254 pJ/bit	4.928
	Size	128 KB		
	Bus Width	512 bit		
	Number	16		
4-1 Pooling Units	Precision	8-bit	0.044 pJ/unit/op	0.1297
	Number	1024		
Accumulation Units	Max Bit	26-bit	9.795 pJ/unit/op	6.4169
	Number	1024		
Activation Units	Precision	8-bit	0.014 pJ/unit/op	0.0134
	Number	1024		
L2 Buffer	Device	SRAM	0.132 pJ/bit	1.95
	Size	16 KB		
	Bus Width	256 bit		
	Number	24		
Chip Total (VGG-16Cov)			7.58 mJ	69
PE Level (432 PEs on Chip)				
Sub-Array	Number	16	25.04 nJ/op	0.101
L1 Buffer	Device	Register	0.064 pJ/bit	0.0073
	Size	4 Kb		
	Bus Width	128 bit		
Adder Tree	Max Bit	16-bit	15.90 pJ/op	0.01
	Number	128		
Output Buffer	Device	Register	0.064 pJ/bit	0.0073
	Size	4 Kb		
	Bus Width	128 bit		
PE Total			0.202 µJ	0.127
Sub-Array Level (16 Sub-Arrays in one PE)				
RRAM array	Size	128*128	0.88 pJ/op	0.0003
	Cell Precision	2-bit		
ADC	Precision	5-bit	22.45 pJ/op	0.0036
	Number	16		
Shift-Add & DFF	Precision	14-bit	1.6 pJ/op	0.0014
	Number	16		
Sub-Array Total			1.565 nJ	0.0063

Fig. 7 shows the latency and energy consumption of interconnect (normalized to the first convolutional layer) and buffers (normalized to the last convolutional layer) along the convolutional layers in VGG-16 [10]. Since for interconnect, the size of weight matrix used to map the synaptic weights (the first layer has the minimum weight matrix) determines the latency and energy consumption, while for the buffers, the input feature sizes (the last layer has smallest input size) determine those. For the baseline design, in principle, the latency and energy consumption should be higher in deeper layer, but there are some drops in layer 3, 5, 8 and 11, latency and energy consumption should be higher in deeper because there are max-pool layers before those layers, which cause 4× decrease of transferring input data, but the results increase dramatically after that, because the weight matrix are doubled and leading to longer transferring distance (e.g. weight matrix size in layer 8 is 2304×512, while in layer 9 is 4605×512). For the proposed design, since the input reuse is maximized, each time only 1/3 of the input as for baseline (kernel size is 3×3) will be transferred from buffer to the nearest PEs, and the input being reused will be transferred among the neighboring PEs through a very short distance simultaneously, thus the bus width will no longer limit the latency that much (as it does in baseline) and the results of interconnect do not vary too much along layers. While the results of buffer vary along layers, because of the limitation of bits that can read out from buffers at each access time. Overall, the novel mapping and data flow can help to save ~65% of latency and ~69% of energy consumption of interconnect and buffers. Table II summarizes the estimation results of the 8-bit RRAM architecture for all the convolutional layers in VGG-16 [10], this design could also extend to XNOR-RRAM [11] for binary neural network.

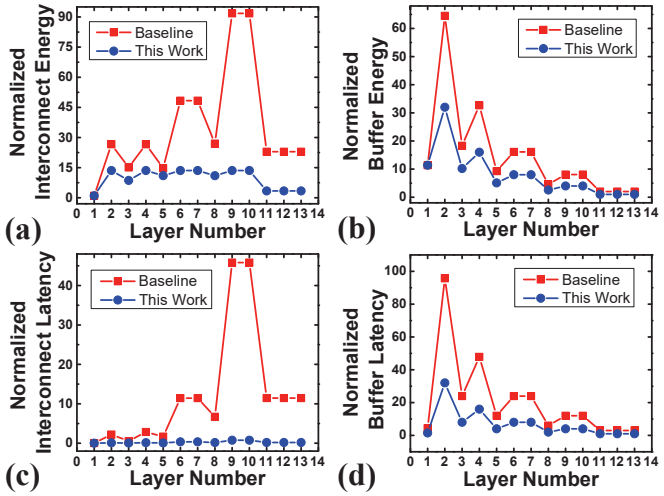

Fig. 7. (a) Normalized energy of interconnect and (b) buffer, (c) Normalized latency of interconnect and (d) buffer for both baseline and this work.

TABLE II. BENCHMARK RESULTS

Architecture	XNOR		8-bit	
Chip Area (mm^2)	12		69	
Hardware Performance	Latency (ms)	TOPs/W	Latency (ms)	TOPs/W
Baseline	1.293	124.2	12.737	4.045
This Work	0.748	149.6	5.882	4.767
Improvement	**1.729×**	**20.4%**	**2.165×**	**17.8%**

V. CONCLUSION

In this paper, we propose an 8-bit RRAM based PIM architecture based on a novel mapping method and data flow which can maximize weight and input data reuse. To analyze the latency and energy saving of interconnect and buffers, we set a baseline which uses conventional mapping method and H-tree routing. We used NeuroSim [8] to estimate the area, latency and energy of VGG-16 [10] benchmark at 32nm, which shows at least ~65% save of latency and energy in interconnect and buffers. This novel mapping and data flow achieve overall ~2.1× speed up and ~17% improvement in energy efficiency, where the 8-bit architecture is ~4.76 TOPS/W, and XNOR-RRAM architecture is ~149.6 TOPS/W.

ACKNOWLEDGEMENT

This work is supported by ASCENT, one of the SRC/DARPA JUMP centers, NSF-CCF-1903951, NSF-CCF-1740225, SRC Contract 2018-NC-2762 and Samsung.

REFERENCES

[1] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1-12.

[2] Y. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127-138, Jan. 2017.

[3] S. Yu, "Neuro-inspired computing with emerging non-volatile memory," *Proc. IEEE*, vol. 106, no. 2, pp. 260-285, 2018.

[4] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose and R. W. Linderman, "Memristor Crossbar-Based Neuromorphic Computing System: A Case Study," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 10, pp. 1864-1878, Oct. 2014.

[5] A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2016, pp. 14-26.

[6] P. Chi et al., "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27-39.

[7] P. Y. Chen and S. Yu, "Partition SRAM and RRAM based synaptic arrays for neuro-inspired computing," *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 2310-2313.

[8] P. Y. Chen, X. Peng and S. Yu, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[9] T Gokmen, M Onen, W Haensch, "Training deep convolutional neural networks with resistive cross-point devices, " *Frontiers in Neuroscience* 11, 538, 2017.

[10] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition, " *International Conference on Learning Representations (ICLR)*, 2015.

[11] X. Sun, S. Yin, X. Peng, R. Liu, J. s. Seo and S. Yu, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," *ACM/IEEE Design, Automation & Test in Europe Conference (DATE)*, 2018, pp. 1423-1428.